PC3

ALGORITMOS Y ESTRUCTURAS DE DATOS

LOCALITY SENSITIVE HASHING - LSH

PRODUCT QUANTIZATION - PQ

OPTIMAL PRODUCT QUANTIZATION -OPQ

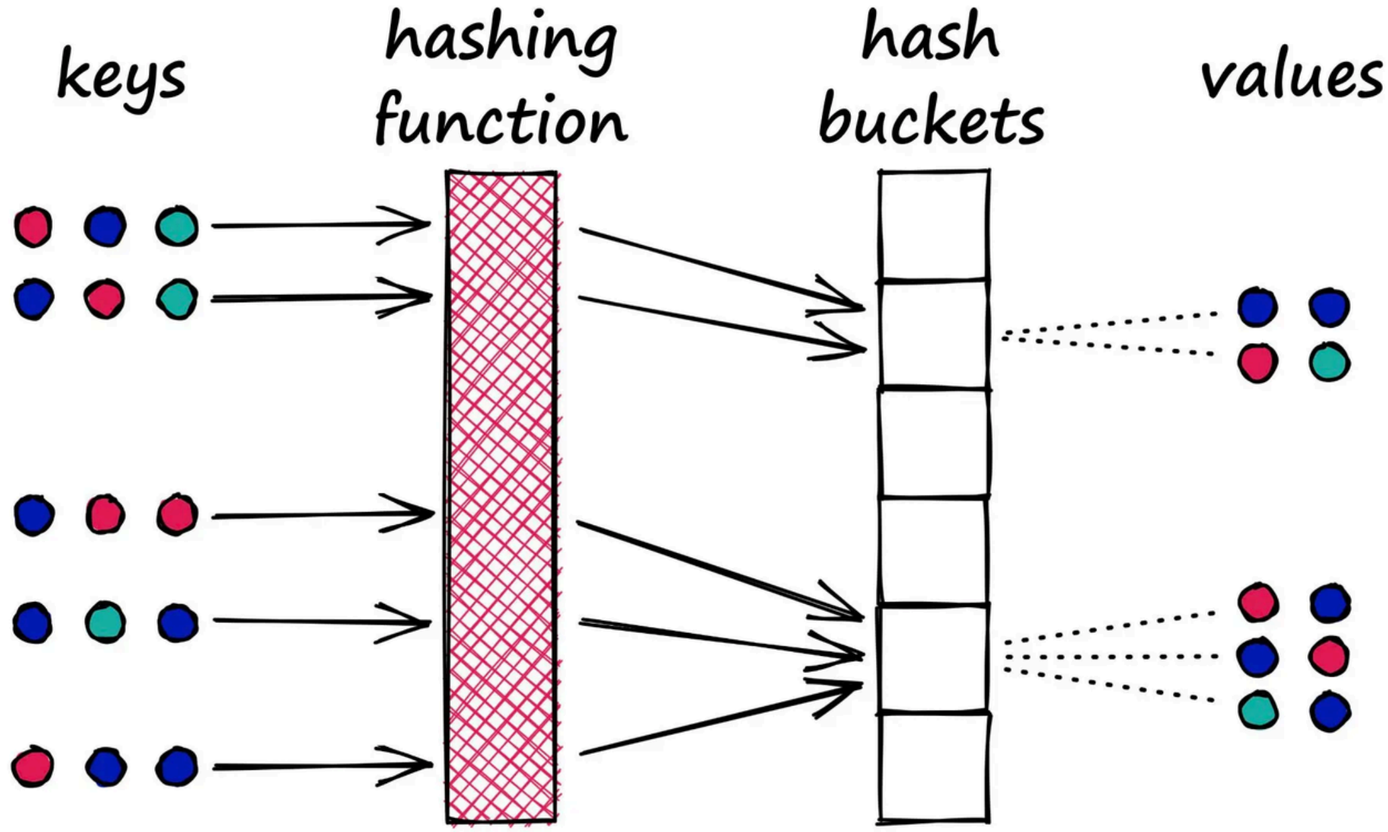
- García Rojas Diego Akira
- López Julcarima Ariana Camila

ÍNDICE

- LSH
- PQ
- OPQ
- Profiling y comparación

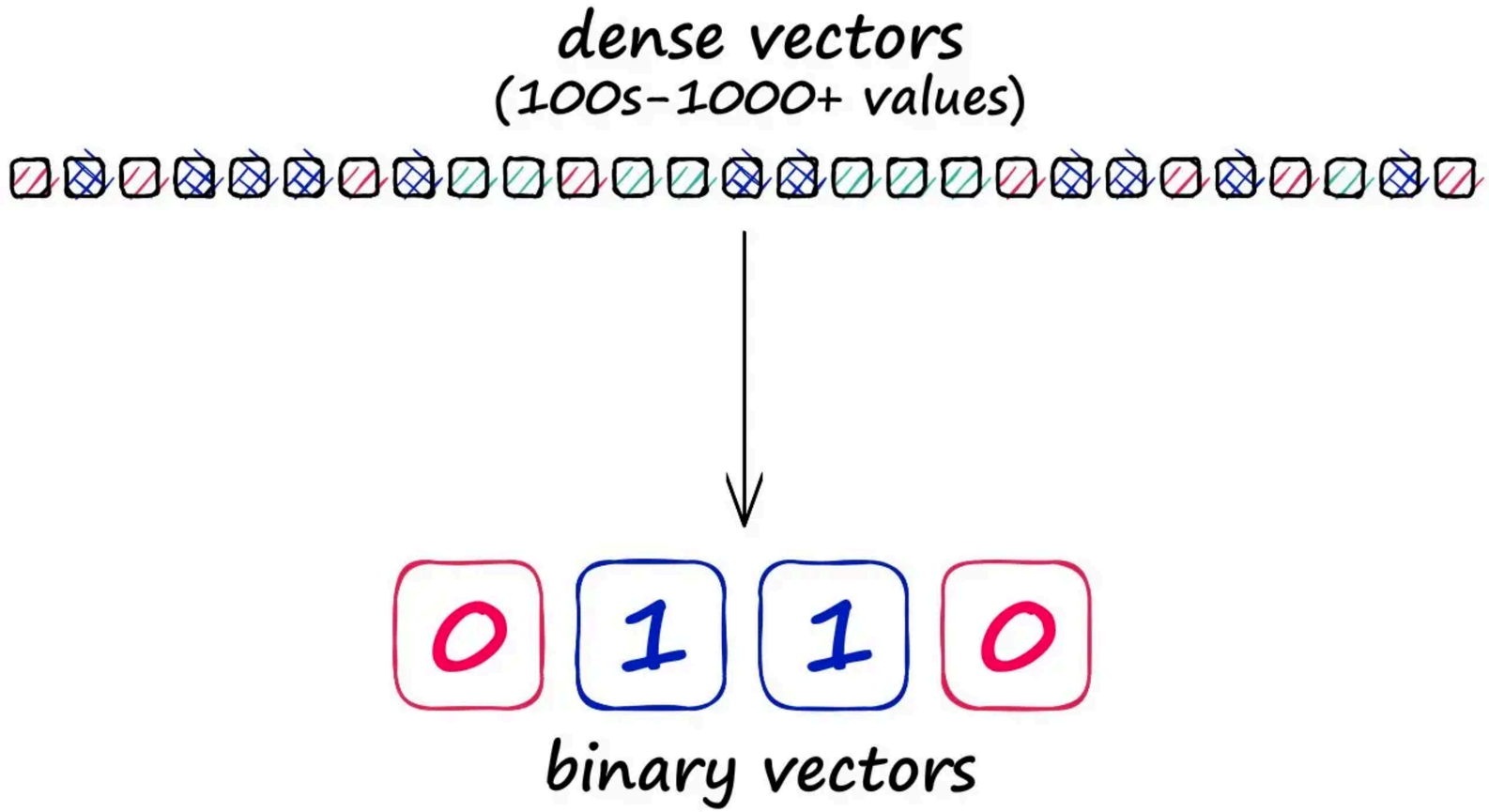
LOCALITY SENSITIVE HASHING - LSH

LSH emplea una función hash sensible a similitud, es decir, si dos puntos están cerca en el espacio original, tienen una alta probabilidad de caer en el mismo bucket después de aplicar LSH.



vectores similares → hash() → buckets similares

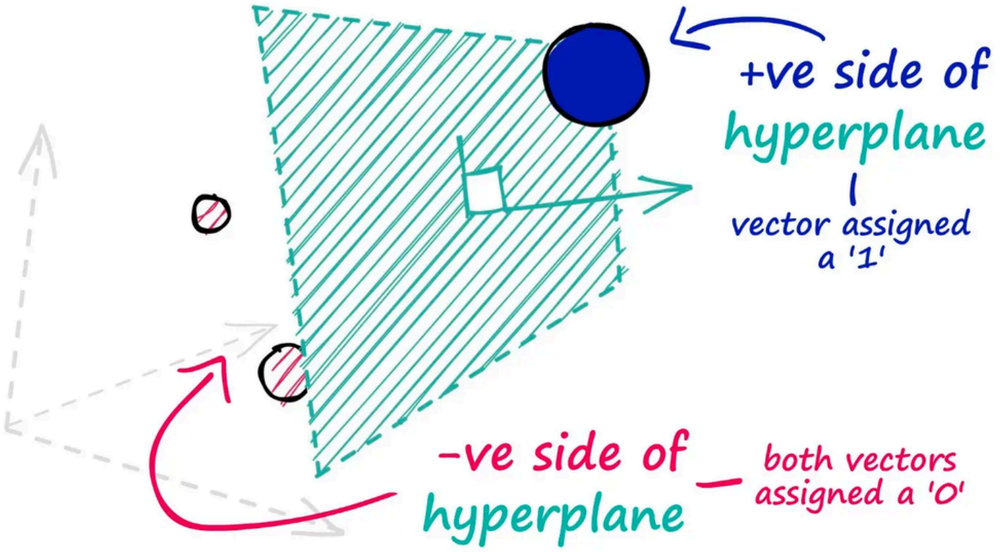
Función Hash: Usando el método de proyecciones aleatorias, reduciremos nuestros vectores de alta dimensión a vectores binarios de baja dimensión. Una vez que tengamos estos vectores binarios, podemos agrupar los vectores en buckets.

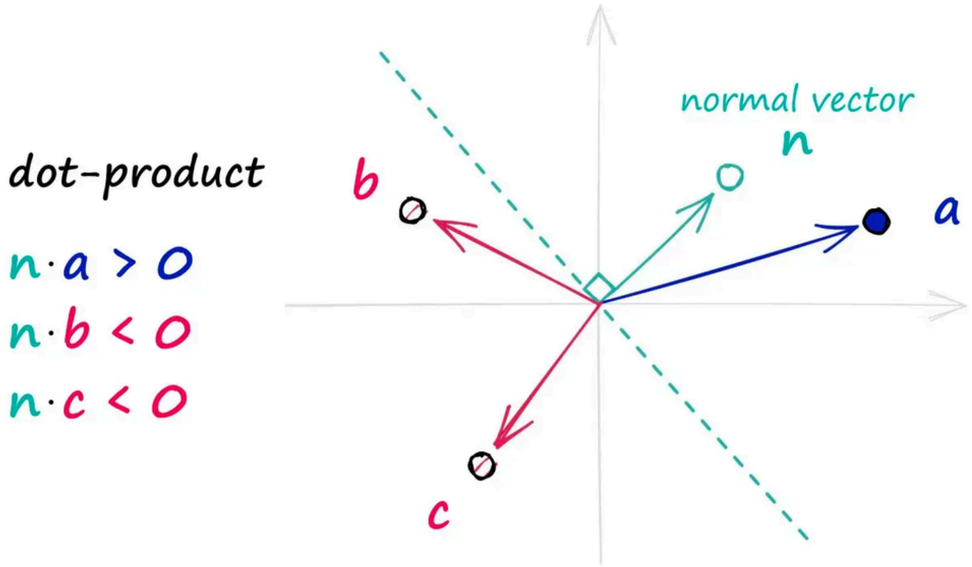


FUNCIÓN HASH: PROYECCIONES ALEATORIAS CON HIPERPLANOS

Cada hiperplano separa el espacio en dos lados. Para cada punto, determinamos en que lado del hiperplano se encuentra.

- Si un punto cae en el lado negativo del hiperplano, se le asigna un 0.
- Si cae en el lado positivo, se le asigna un 1.



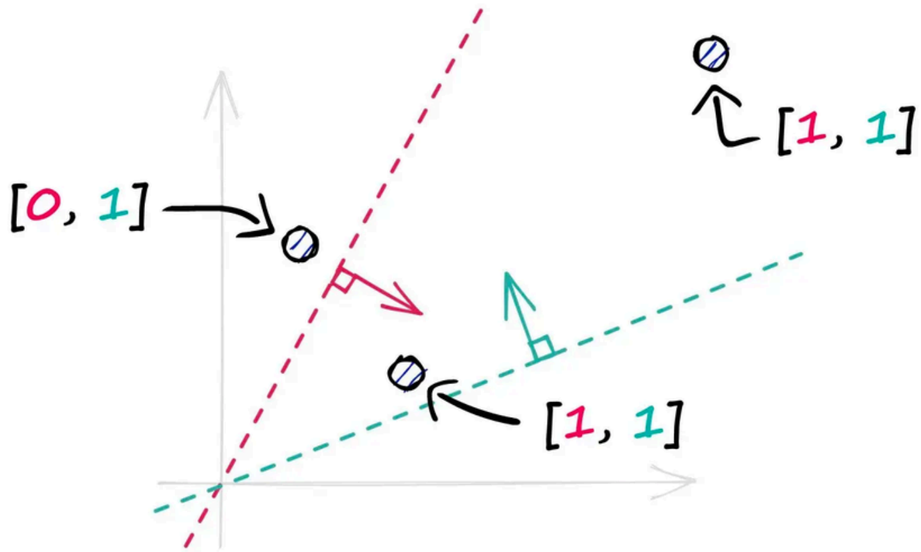


Para determinar en qué lado del hiperplano se encuentra un punto, se calcula la proyección del punto sobre el vector normal al hiperplano

- Si el resultado es positivo, el punto está en el lado positivo del hiperplano (misma dirección).
- Si es negativo, está en el lado negativo (dirección opuesta).

FUNCIÓN HASH: PROYECCIONES ALEATORIAS CON HIPERPLANOS

Los hiperplanos tienen como función principal dividir el espacio en regiones. Al proyectar un punto sobre varios hiperplanos, podemos determinar en que región cae cada uno, lo que nos permite agrupar puntos similares en un mismo bucket.



Proyectamos un vector sobre k hiperplanos

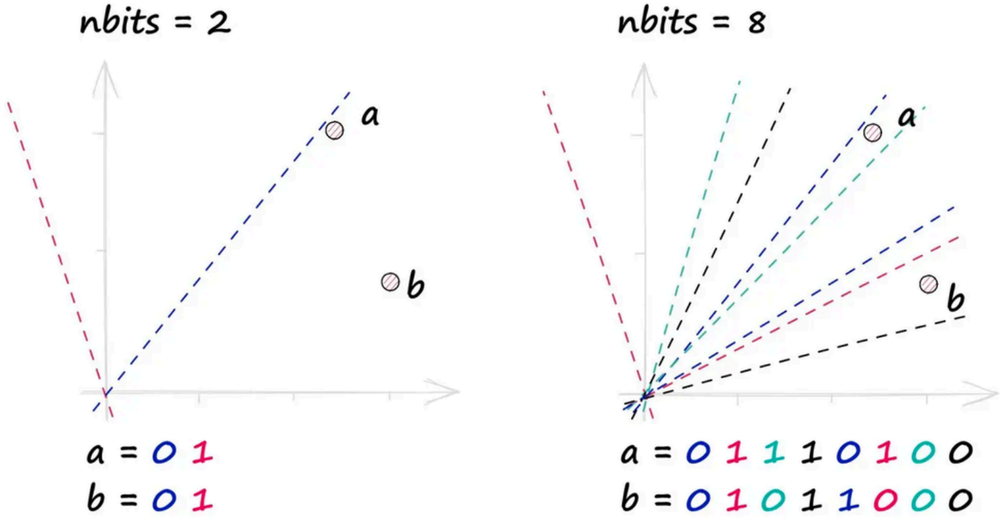
vector → h1(vector) → indice 1 → 0 - 1
vector → h2(vector) → indice 2 → 0 - 1
...
vector → hk(vector) → indice k → 0 -1

Donde:

- hi(vector): proyección del vector sobre el hiperplano i

vector → h(vector) → [vector binario de dim k]

La información de los k indices determinara el bucket al cual pertenece el vector de entrada



Para obtener los puntos candidatos mas cercanos a un punto de consulta, vamos a buscar también buckets vecinos al bucket original en el que cae dicha consulta.

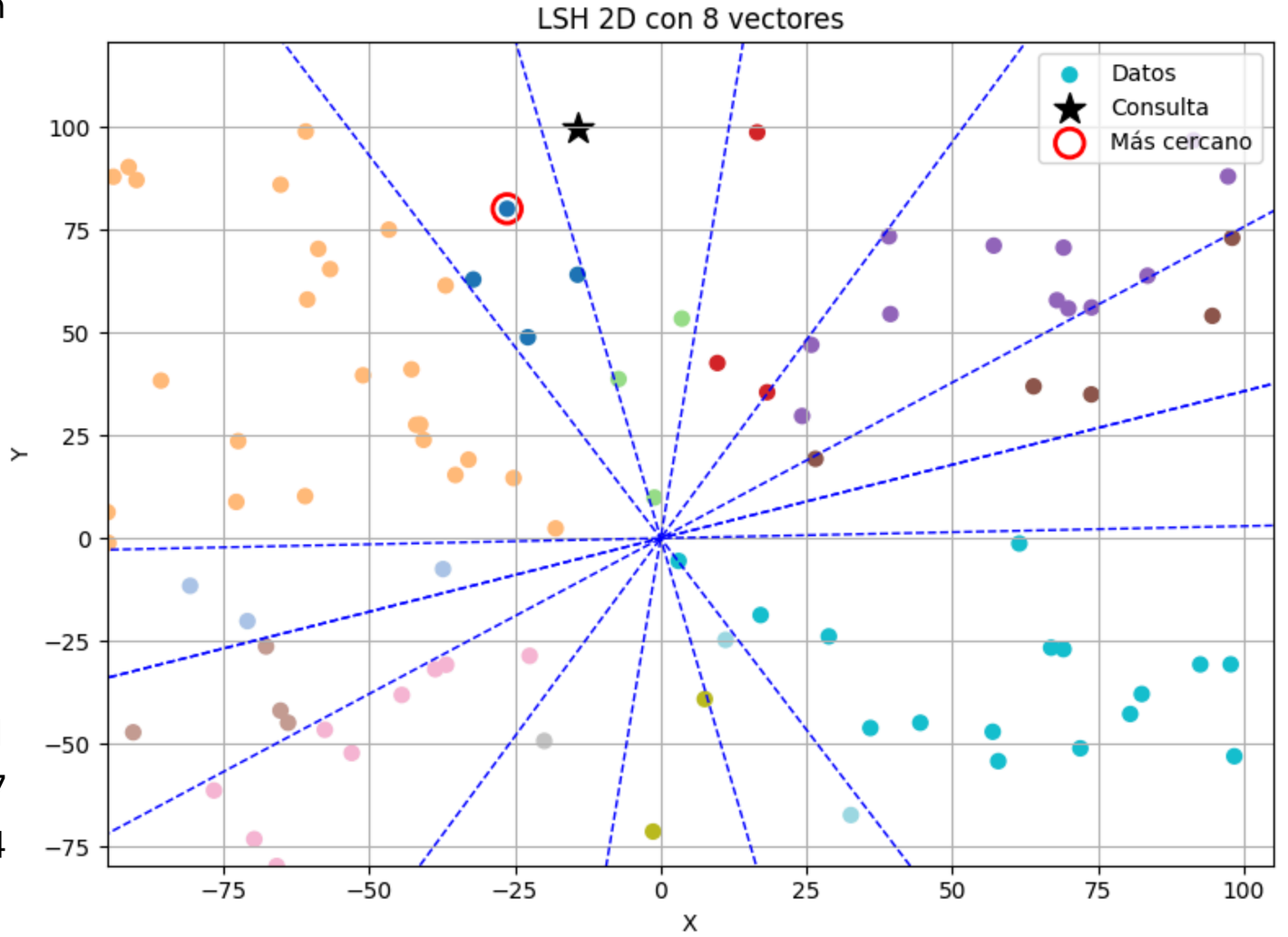
Se generan buckets vecinos alterando algunos bits del vector binario del query, según un radio, ya que puntos similares pueden haber caído en buckets distintos por mínimas diferencias.

Por ejemplo:

- query: [1, 0, 1]
- radio máximo: 1

Cambio en posición 0: [0, 0, 1] → binario 001 → bucket 1
Cambio en posición 1: [1, 1, 1] → binario 111 → bucket 7
Cambio en posición 2: [1, 0, 0] → binario 100 → bucket 4

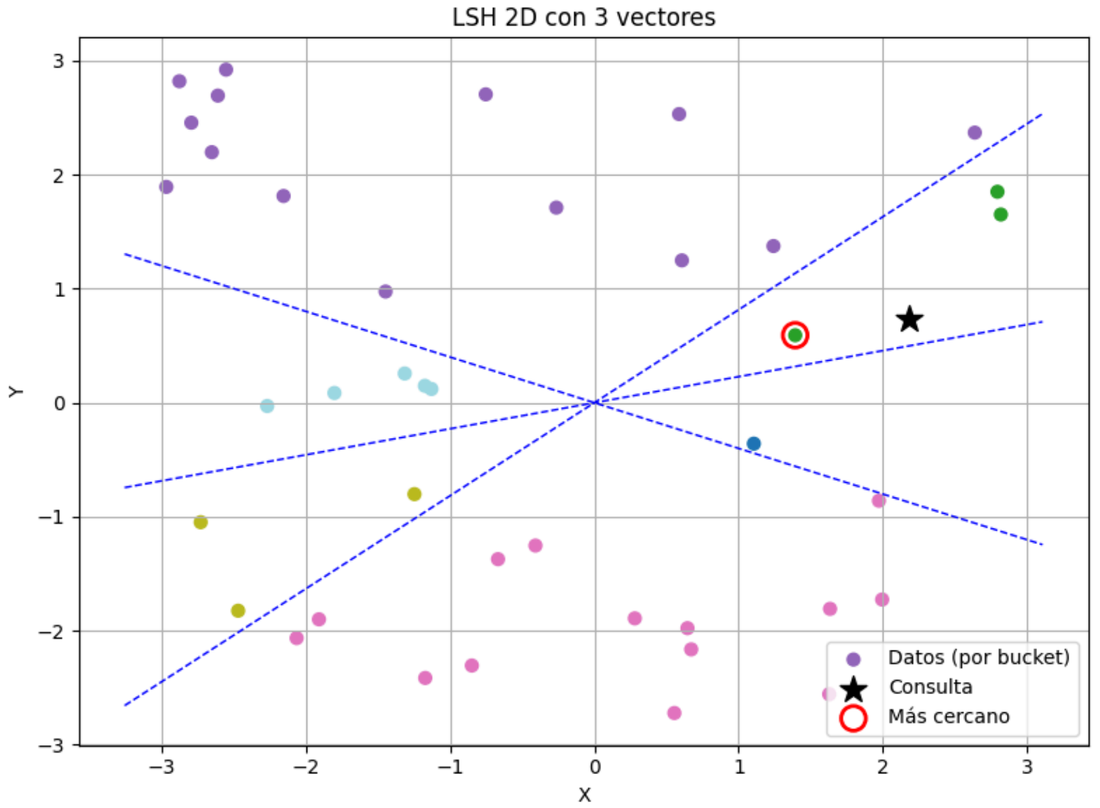
Si existen, se agregan como candidatos a ser vecinos del punto de consulta.

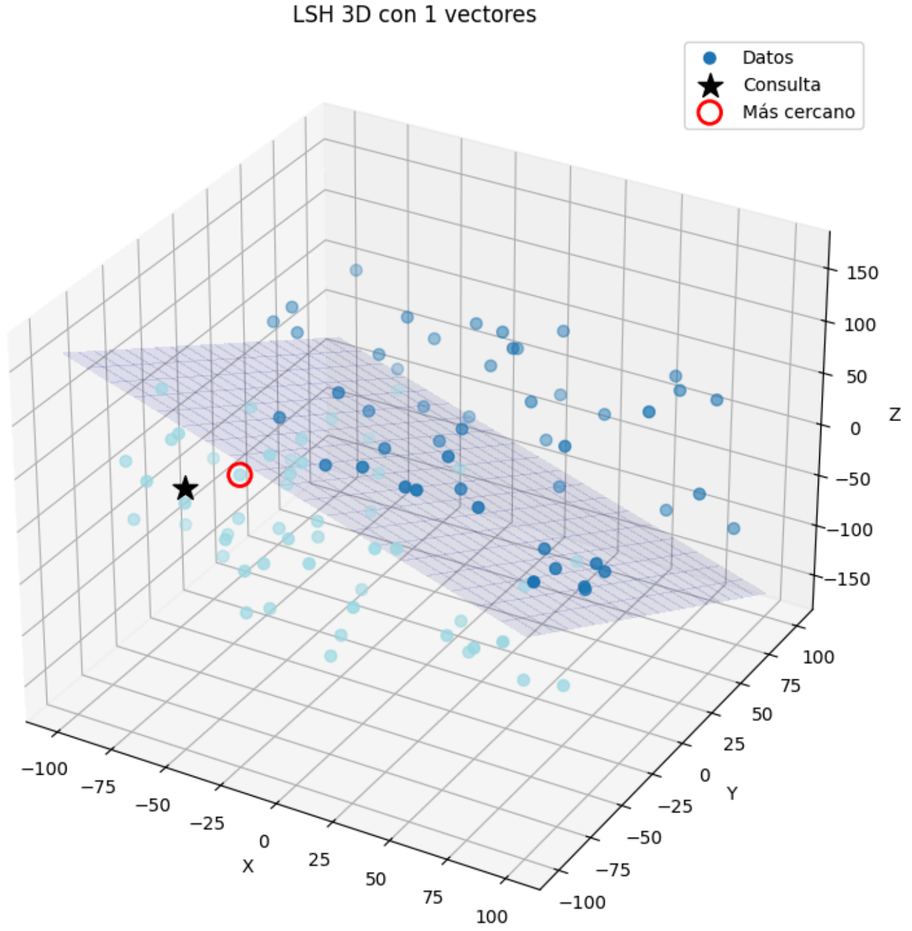


Si solo buscáramos en el bucket original, podríamos perder posibles vecinos cercanos que, por un cambio mínimo en el hash, cayeron en otro bucket.

LOCALITY SENSITIVE HASHING

Búsqueda de vecinos

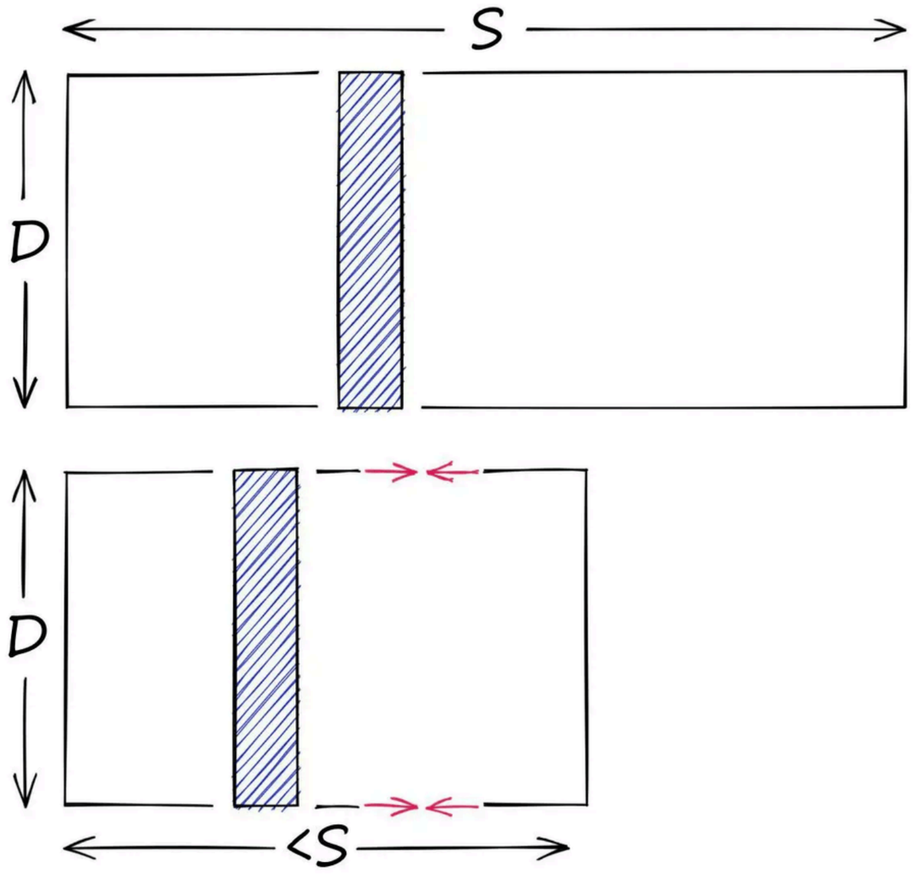




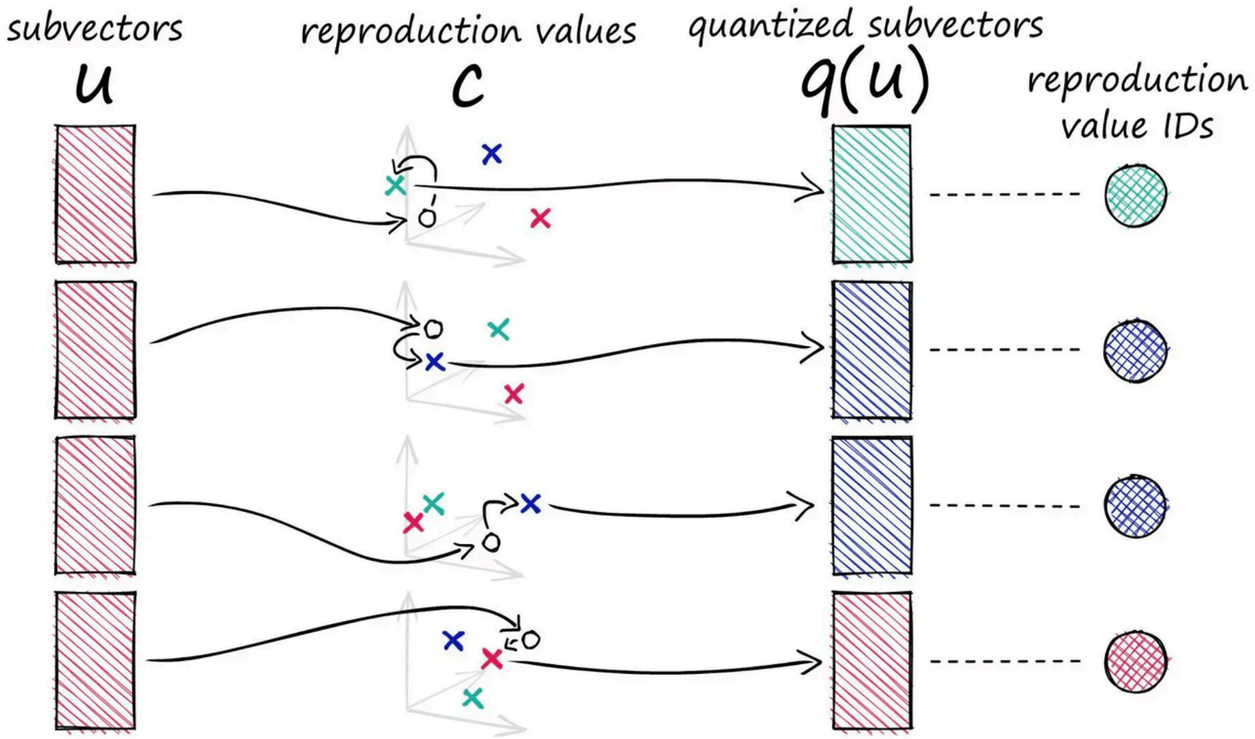
PRODUCT QUANTIZATION

- Quantization: La cuantización es el proceso de reducir el tamaño de un conjunto de datos reduciendo el universo de valores que puede tomar.

- Product: Los datos resultantes son el producto cartesiano de un conjunto de valores discretos y conocidos.



PRODUCT QUANTIZATION

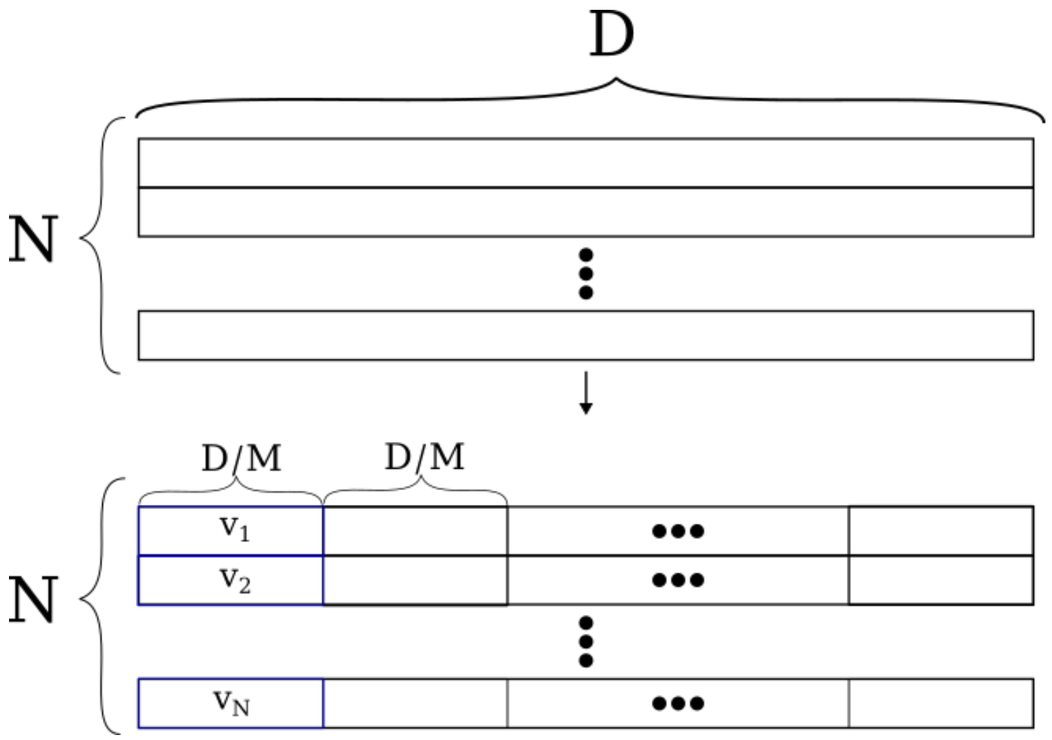


Dado un conjunto de vectores, se busca reducir el tamaño que ocupan mediante los siguientes pasos:

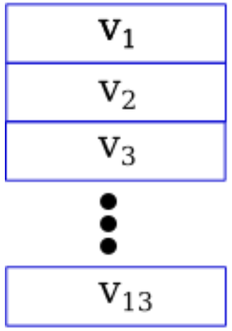
1. Se dividen los vectores en subvectores de igual dimensión.
2. Lo subvectores definen un subespacio para los que son clusterizados, obteniendo un conjunto de k de centroides para cada subespacio. Cada conjunto de centroides es denominado codebook.
3. Los subvectores son representados como el índice de su centroide más cercano en el codebook.
4. Los vectores originales son representados como arreglos de índices.

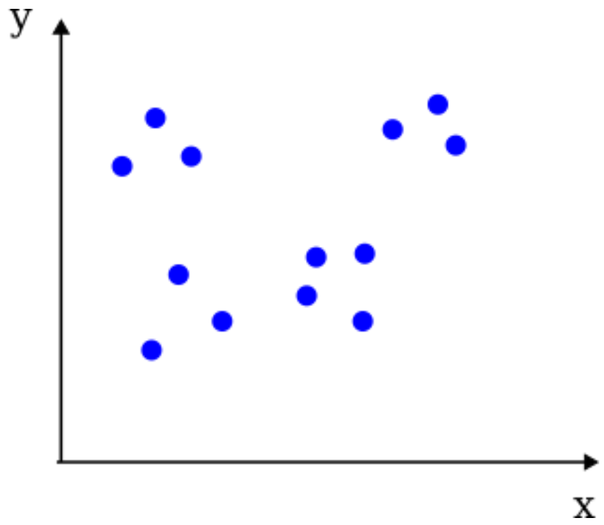
PRODUCT QUANTIZATION

1. Se definen los subvectores a partir de un conjunto de vectores



- Asumiendo que se tienen N = 13 vectores y subvectores de dimensión D/M = 2, se tiene el siguiente subespacio:

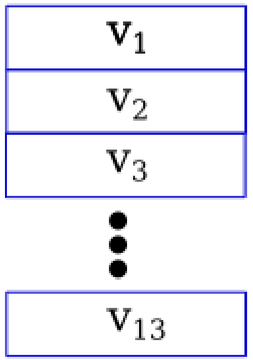


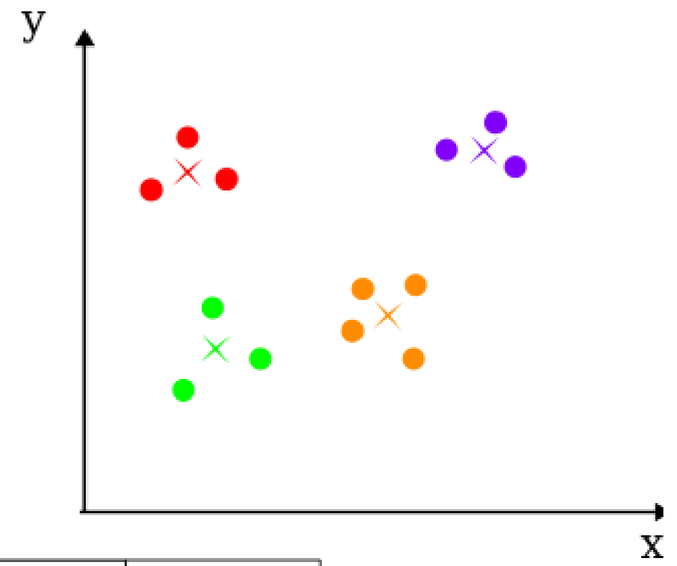


PRODUCT QUANTIZATION

3. Se clusterizan y se definen los codebooks.

- En la implementación se usa el algoritmo KMeans implementado en scikit-learn.





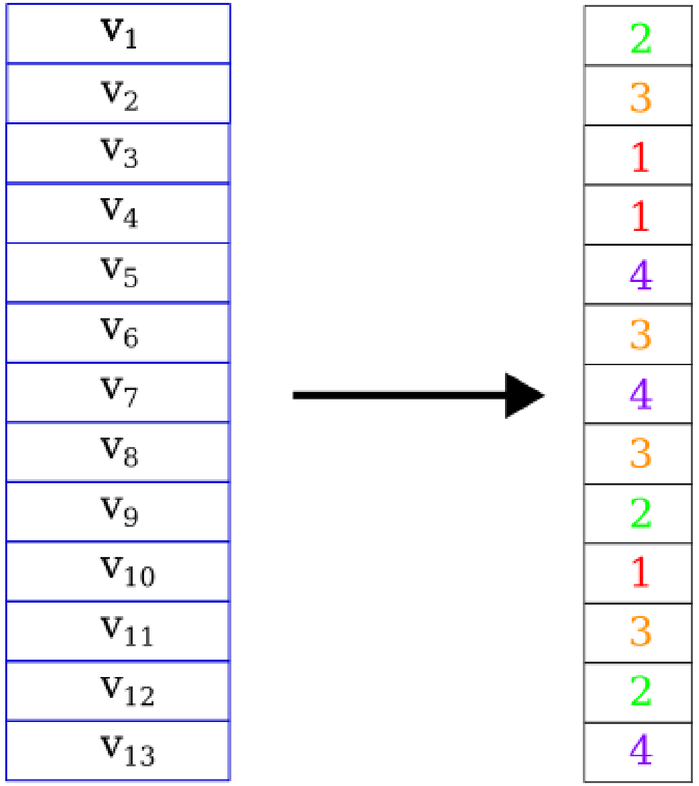
Codebook	Índice
C_1	1
C_2	2
C_3	3
C_4	4

PRODUCT QUANTIZATION

3.Los subvectores son representados como el índice al centroide más cercano en el codebook.

4. Se repite este proceso para cada subespacio.

- En este ejemplo, se reduce el espacio de 13 subvectores de dos dimensiones a tan solo 13 enteros o bytes (cuantización).
- Los vectores originales se representan como el producto cardinal de los índices que en su lugar representan centroides (producto).



PRODUCT QUANTIZATION

Búsqueda de vecinos

- La búsqueda de vecinos se realiza mediante Asymmetric Distance Computation (ADC):
  - El vector de consulta mantiene su precisión completa.
  - Los vectores almacenados se mantienen cuantificados.

PRODUCT QUANTIZATION

Búsqueda de vecinos

Vector de consulta

q

q_1	q_2	q_3

Codebooks		
Codebook 1	Codebook 2	Codebook 3
C_{11}	C_{21}	C_{31}
C_{12}	C_{22}	C_{32}
C_{13}	C_{23}	C_{33}
C_{14}	C_{24}	C_{34}

Distancia a codebooks		
Codebook 1	Codebook 2	Codebook 3
$d(q_1, C_{11}) = D_{11}$	$d(q_2, C_{21}) = D_{21}$	$d(q_3, C_{31}) = D_{31}$
D_{12}	D_{22}	D_{32}
D_{13}	D_{23}	D_{33}
D_{14}	D_{24}	D_{34}

1. Se divide el vector de consulta en subvectores de la misma dimensión que los centroides.
2. Se calcula la distancia de cada subvector a cada centroide de su subespacio correspondiente.

PRODUCT QUANTIZATION

Búsqueda de vecinos

3. Para hallar la distancia desde el vector de consulta a los vectores almacenados, solo se suma las distancias a las que apuntan sus códigos.
4. Se recuperan las distancias mínimas para recuperar los vecinos.

Distancia a codebooks		
Codebook 1	Codebook 2	Codebook 3
D_{11}	D_{21}	D_{31}
D_{12}	D_{22}	D_{32}
D_{13}	D_{23}	D_{33}
D_{14}	D_{24}	D_{34}

v

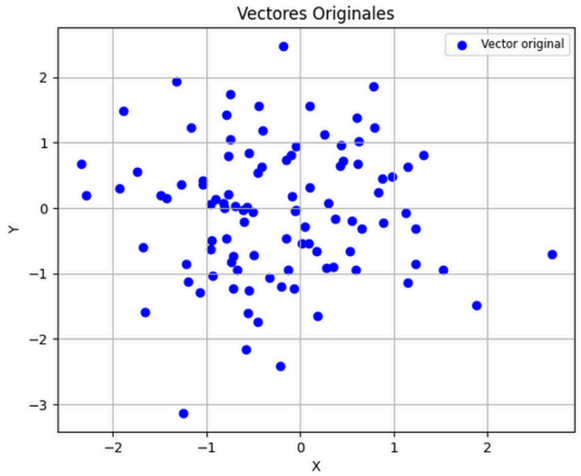
4	2	1

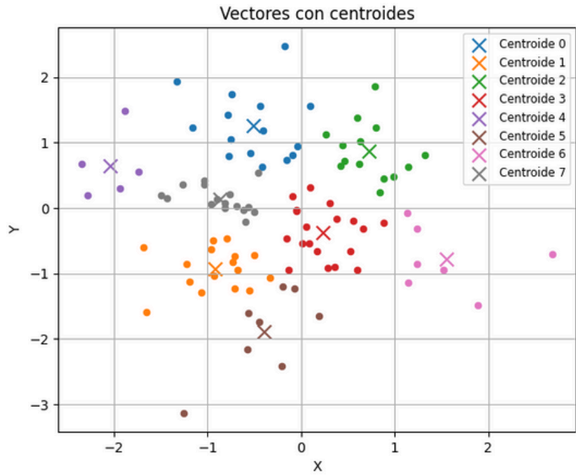
Distancia aproximada de q a v

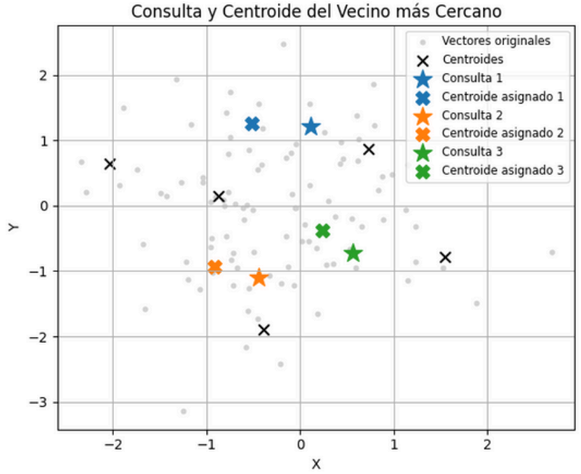
$$D_{14}+D_{22}+D_{31}$$

PRODUCT QUANTIZATION

Búsqueda de vecinos



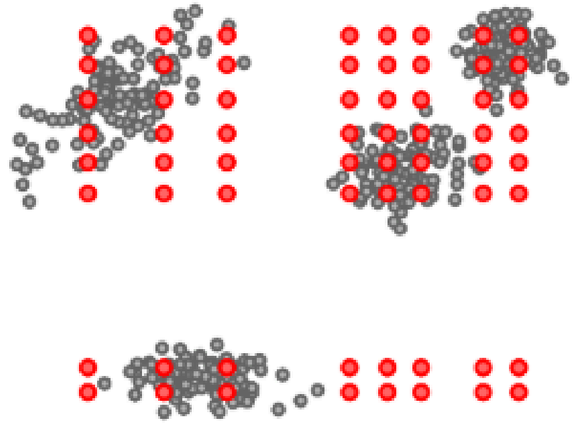




- Solo se calculan (Número de centroides)*(Número de codebooks) distancias → Mucho más rápido que fuerza bruta.
- En este ejemplo se tienen 100 vectores de dos dimensiones codificados a centroides de dos dimensiones.
  - La búsqueda de vecinos solo recupera el centroide más cercano.

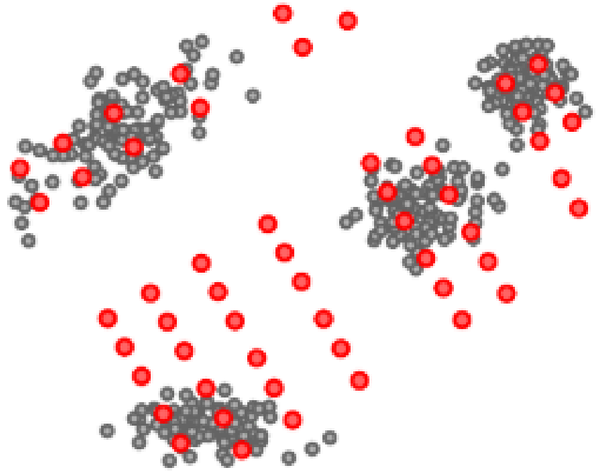
OPTIMIZED PRODUCT QUANTIZATION

El método OPQ es una variación de PQ, que aplica una rotación previa a los datos.



(b) PQ

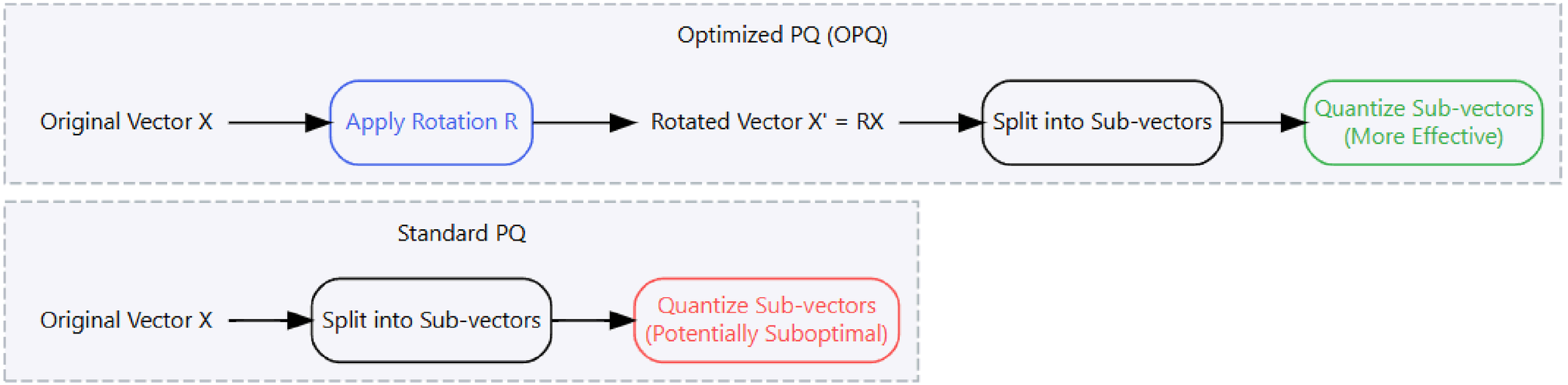
En PQ, la división en subespacios es fija y alineada con el eje vertical, sin embargo, como los datos no están alineados con esos ejes, los centroides terminan lejos de los puntos reales, lo que genera mayor error.



(c) OPQ

En cambio, OPQ primero rota los datos para que esa división por subespacios se alinee mejor con la distribución real de los datos. En consecuencia, los centroides siguen una forma diagonal, adaptándose mucho mejor a la estructura.

OPTIMIZED PRODUCT QUANTIZATION



El objetivo de OPQ es rotar los datos de manera óptima antes de aplicar Product Quantization, con el fin de minimizar el error de reconstrucción, entonces el primer paso es encontrar una rotación óptima.

ENTRENAMIENTO ITERATIVO EN OPQ

OPQ busca una matriz de rotación ortogonal R que minimice el error de reconstrucción al aplicar Product Quantization sobre los vectores rotados

- Se empieza con R=I por lo que inicialmente OPQ se comporta igual que PQ.
- Se calcula Xrot = X · R
- Se aplica PQ sobre Xrot usando k-means para encontrar centroides por subespacio.
- Se reconstruyen los vectores desde sus códigos y se compara contra los datos originales.
- Se busca una nueva matriz R que minimice el error entre los originales y los reconstruidos.

$$R = \arg\min_{R} \|X - (X_{\text{rot}} \cdot R^{\top})\|^2$$

Y se resuelve eficientemente usando SVD sobre

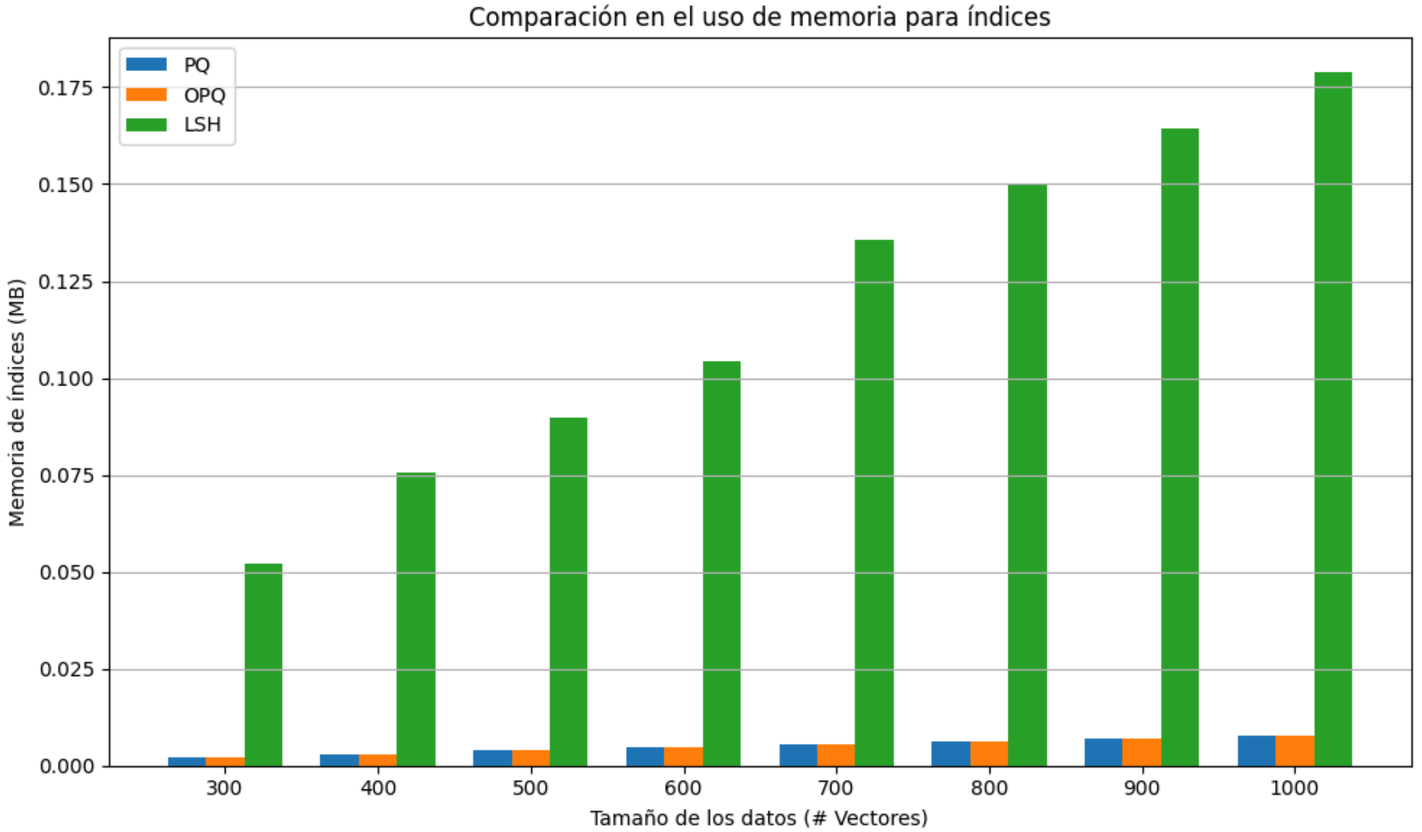
$$X^{\top} \cdot \hat{X}$$

como resultado obtenemos R una matriz ortogonal que minimiza el error cuadrático de alineación.

PROFILING

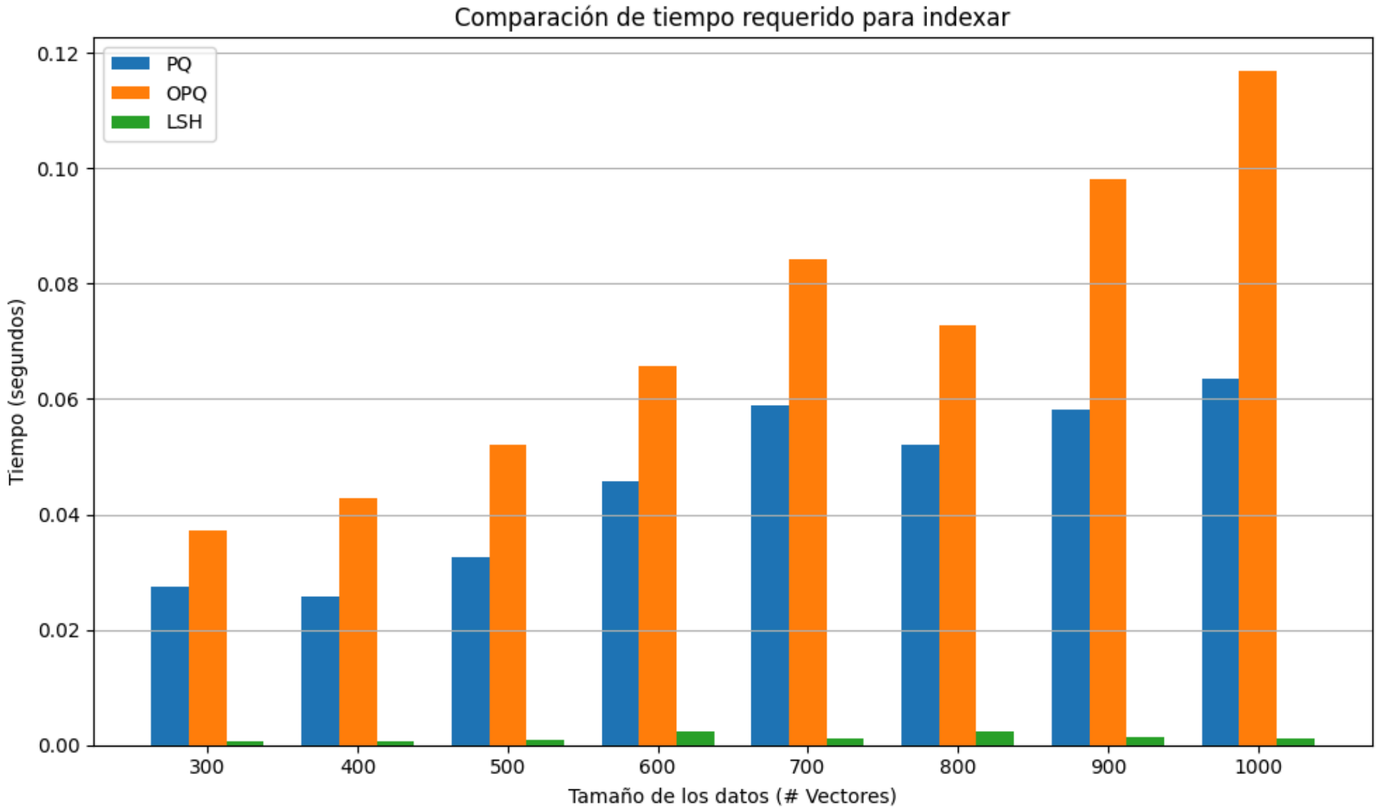
Memoria

- PQ y OPQ solo almacenan índices → Optimizado para almacenamiento.
- LSH almacena un diccionario → Aumenta memoria con más datos.



PROFILING

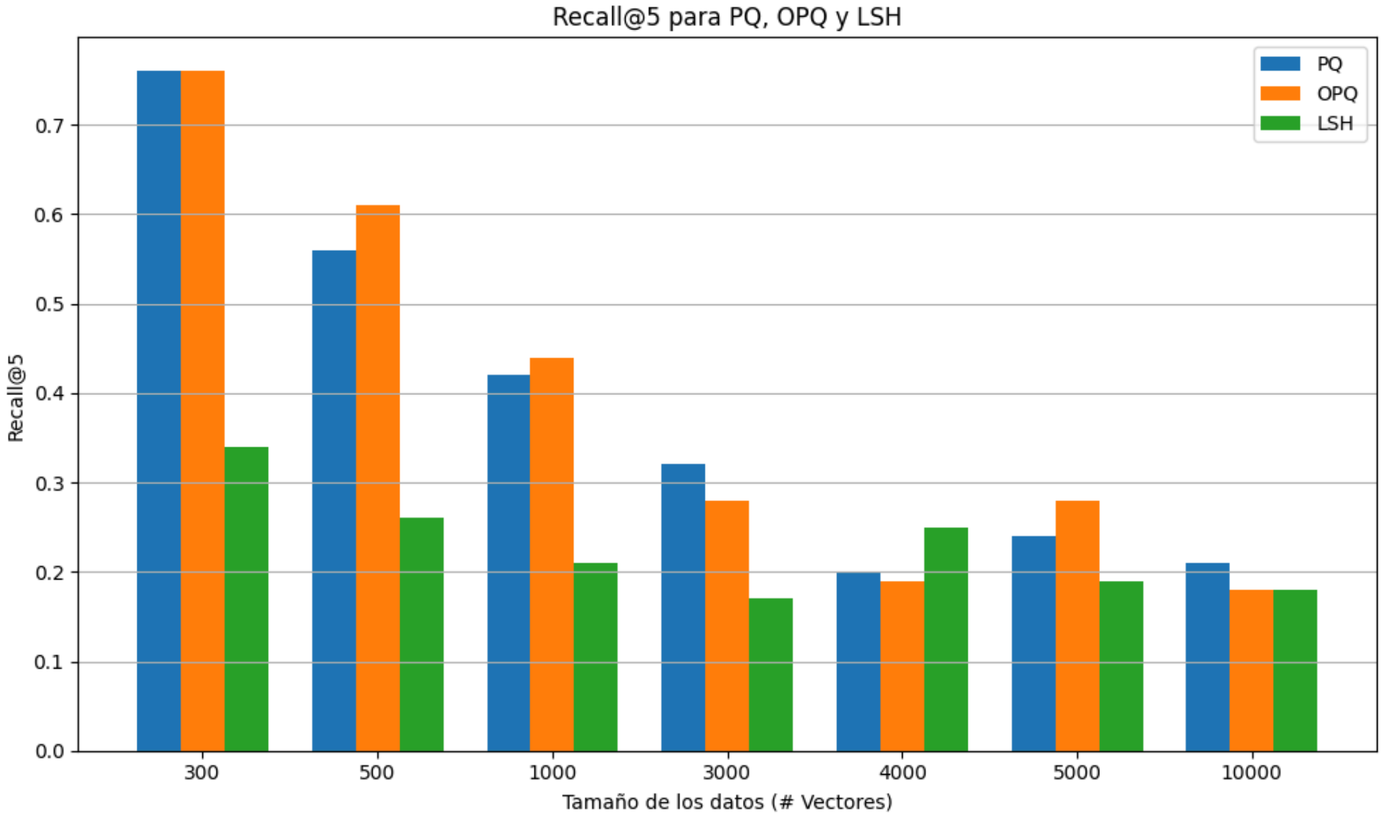
Tiempo



- PQ y OPQ requieren calcular N distancias múltiples veces y hallar centroides.
  - OPQ requiere multiplicaciones de matrices.
- LSH solo requiere hashes.

PROFILING

Recall@5



- PQ y OPQ son considerablemente precisos para pocos datos, pero son limitados por muchos datos.
  - OPQ es más preciso que PQ.
- LSH no es conveniente para pocos datos pero puede sobrevellevar grandes datos mejor.

GRACIAS